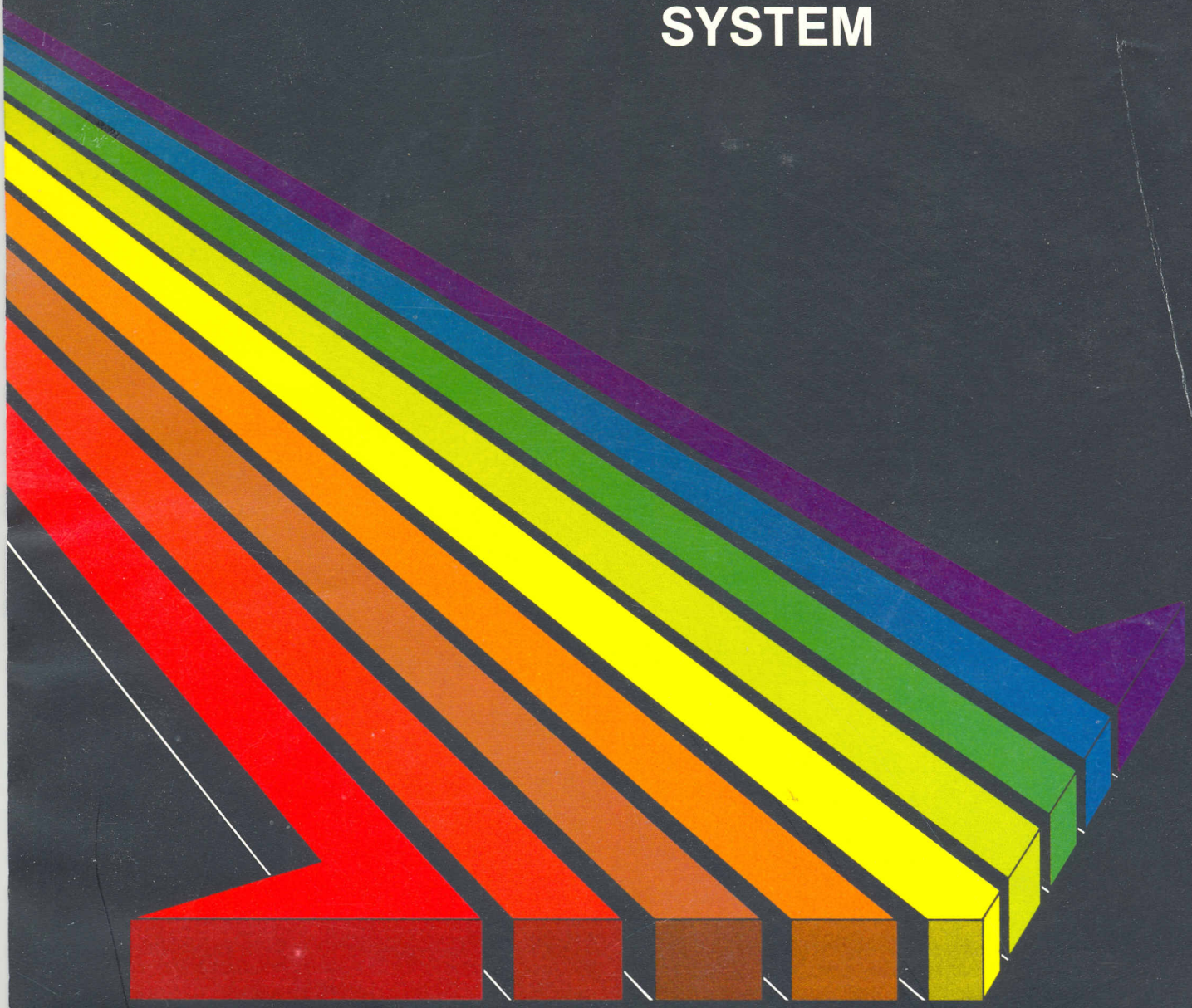
intel

HOW TO SELECT A VLSI OPERATING SYSTEM



How To Select A VLSI Operating System

Contents

January 1982

intel®

ARTICLE
REPRINT

AR-194

November 1981

Choosing a VLSI-Compatible System

Peter Palm
Mini-Micro Systems

Order Number: 210341-001

OPERATING SYSTEMS

Choosing a VLSI-compatible system

PETER PALM, Intel Corp.

VLSI offers expanded modularity for operating systems; the intended application will determine the most appropriate system

By integrating more functions into silicon, very large-scale integrated (VLSI) circuitry lowers the cost of μcs and improves their performance and ease of use. But the advent of VLSI raises a question: which μc operating systems enable a user, especially an OEM, to take advantage most quickly of these hardware improvements? This question can be answered by examining the characteristics of available μc operating systems in the light of VLSI advances.

An operating system's intended application is a key consideration in its selection. No operating system is ideal for every application and customer. For this reason, designers tend to optimize operating systems for specific types of applications.

Microcomputer operating systems fall into two categories: systems optimized for efficient development of new software and those optimized for efficient execution of existing software (Fig. 1). Development-oriented systems tend to sacrifice performance to ease of use, while execution-oriented systems make the opposite trade-off. It is tempting to characterize the two types of systems as either end-user-oriented (execution) systems or OEM (development) systems. But such a categorization is inadequate because many end users are concerned with software development, and many OEMs are concerned with software execution.

Development- versus execution-oriented

In the 8-bit μc world, CP/M and Intel's ISIS (Intellec development system) operating systems are good examples of products targeted at efficient software development. In the fast-growing 16-bit μc segment, ISIS (Series III) and the UNIX operating system and derivatives, such as XENIX, fill the development need. Software developed with one of these operating systems is often executed on another machine running an execution-oriented operating system. As more development throughput is required, such as compilations per hour, users move from single-terminal, single-task systems (such as CP/M) to multiterminal systems (such as UNIX or MP/M) and to multiprocessor development systems (such as Intel's NDS-1).

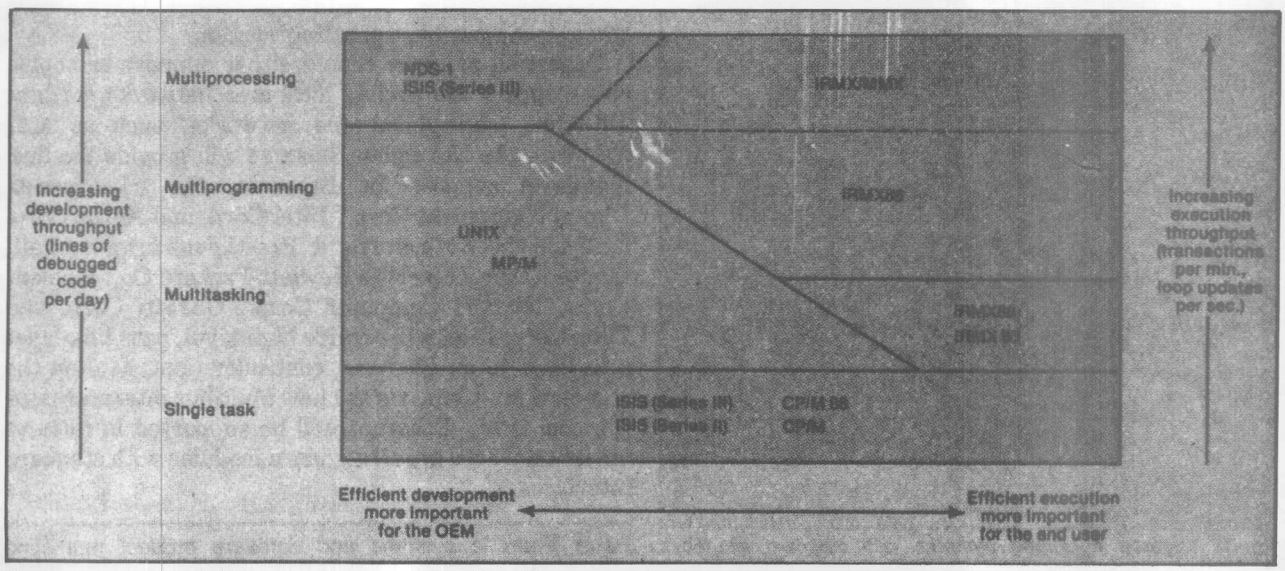


Fig. 1. μC operating systems *are design-optimized for efficient execution for end users (right) or for efficient program development for OEMs (left). They range from single-task, single-terminal systems (bottom) to multiprocessing systems (top).*

An operating system's intended application is key in its selection.

Most μc systems are execution-oriented and are often most critical for OEMs who must stay competitive in price and performance. Rapid program development is usually secondary to efficient software execution. An OEM can gain a significant competitive advantage by using an operating system that provides faster execution times, less expensive investment, ease of use and the ability to be upgraded.

Operating system vendors, including Intel, have designed a range of products for different needs. CP/M, for instance, is a logical candidate for 8-bit μc applications, in which only single-task execution is required. For 16-bit applications, multiprogramming-type operating systems, such as MP/M, iRMX 86 and iRMX 88, are more appropriate because they take advantage of the 16-bit processor's power. An OEM for whom real-time execution is important might consider iRMX 86. If background program development is crucial, MP/M might be a better choice. For highest performance, users should consider multiprocessing operating systems, such as iRMX/MMX800.

Other selection criteria

Although the intended application is paramount, other considerations are important in selecting an operating system. The overriding factor in light of current trends is probably the ability of a system to keep pace with the impact of VLSI on μc performance and cost. This consideration entails several selection criteria, including transferability, multiprocessing architecture, configurability and interfacing to modules.

First, an operating system should be easy to transfer—at least in part—into VLSI silicon. Putting an operating system into EPROM, for example, can improve speed and lower cost. Vendors should state which operating system functions will be integrated into silicon, and when.

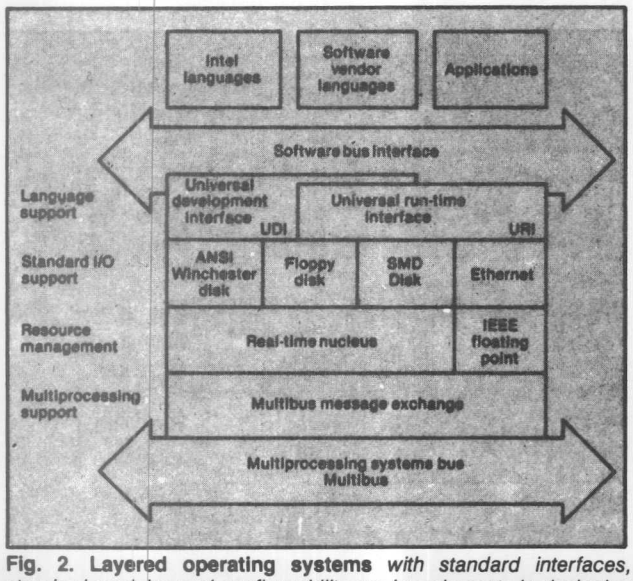


Fig. 2. Layered operating systems *with standard interfaces, standard modules and configurability are key elements in designing μc operating systems to take maximum advantage of lower cost, higher performance VLSI.*

An operating system should provide support for multiprocessing, and VLSI has made multiple-μc systems practical. OEMs should look for operating systems with multiprocessor architectures or other features that ease the move to multiprocessing. Such systems typically include fast context switching, task-to-task communication, synchronization and memory message passing features. For example, the new iMMX800 Multibus message-exchange software allows 8- and 16-bit, master-to-master and master-to-intelligent-slave single-board computers to multiprocess loosely on the Multibus multiprocessor bus.

Modularity is another important criterion in selecting an operating system. The iRMX 86, for example, consists of layers of modules that can easily be moved into silicon as required (Fig. 2). This modular design has enabled Intel to develop a new component dubbed 80130 Operating System Firmware (iOSF) that integrates a timer, an interrupt controller and the iRMX 86 kernel.

An operating system should include standard interfaces to modules. For example, iRMX 86 uses a standard object-oriented format for interfaces to jobs, tasks and message primitives. At a higher layer, iRMX 86 offers standard device-independent interfaces to drivers for the new 8089/8272-based Winchester- and floppy-disk controllers and other device controllers. The interface itself eventually will move into silicon. Only standard interfaces will allow this to occur.

An operating system should also provide industry-standard interfaces to popular program-development languages, such as Intel's universal development interface (UDI) and universal run-time interface (URI). Intel's new UDI/URI-compatible languages, FORTRAN 86, Pascal 86, PLM/86 and ASM 86, can run on any UDI/URI-compatible operating system.

Operating systems should either support or should soon support the leading local-area networks, such as Ethernet, and global-area networks, such as X.25 2780/3780. In November, iRMX 86 will provide the first high-level support for Ethernet, the tri-corporate Digital Equipment Corp., Intel Corp. and Xerox Corp. local-area network standard. Prestigious firms committing to Ethernet include Hewlett-Packard Co., Siemens Corp., Nixdorf Computer Corp., Olivetti Corp. and Zilog Corp. Intel will provide high-level, data-link-layer interfaces to an Ethernet controller (iSBC 550) on the standard Multibus, via the new Multibus interprocessor protocol (MIP). Ethernet will be supported in iRMS 86 via iMMX. These are all standard modules with standard interfaces. ■

Peter Palm is systems and software product marketing manager, OEM Microcomputer Systems Operation, Intel Corp., Hillsboro, Ore.

intel®

ARTICLE
REPRINT

AR-195

December 1981

Microcomputer Operating System Trends

Peter I. Wolochow
Digital Design

Order Number: 210341-001

SOFTWARE

Microcomputer Operating System Trends

advances in VLSI and other technologies force the development of advanced operating systems

As the use of μCs expands, demand for high-level languages and human machine methods of interfacing will expand in 1982 and beyond. One major component of the expanding demand concerns μC operating systems.

by Peter I. Wolochow

An operating system performs resource management and human-to-machine translating functions. Technically, it is just another computer program — a series of instructions that tell the machine what to do under a variety of conditions. Major operating system functions include management of memory, I/O peripherals, and the central processor.

When computers were first developed, the programs (or instructions) were entered into the machine each time a particular job was started. Only with the ability to store a program in the computer's memory, over 30 years ago, did the concept of computers as we know them today truly emerge. The ability to store a program meant that a computer could perform repetitive tasks while the operator had only to enter information upon which calculations were performed.

Once it became possible to store programs, the development of operating systems began. Operating systems were stored in the computer's memory, and provided the user with a bank of stored computer instructions that could be used with a number of different application programs.

Over the years, operating systems have evolved to the point where they have three main purposes. First, they provide clear, consistent, and easily understood guidelines to users concerning how the machine works. A sub-objective is to provide an easier, more "friendly" human interface to the μC. Second, they perform initialization and start-up functions "automatically" so that — to the user — the machine is ready to perform its basic functions. This initialization function originally included only initial start-up, but now often includes methods to recover from errors in both hardware and software. Third, they provide efficient machine and storage resource management so that different users or programs can

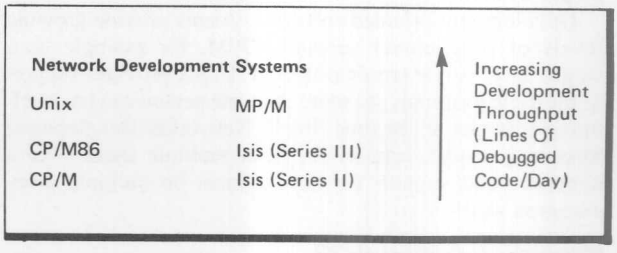


Figure 1: These microcomputer operating systems are primarily for efficient program development.

Peter I. Wolochow is with the OEM Microcomputer Systems Div. of Intel Corp, Hillsboro, OR.

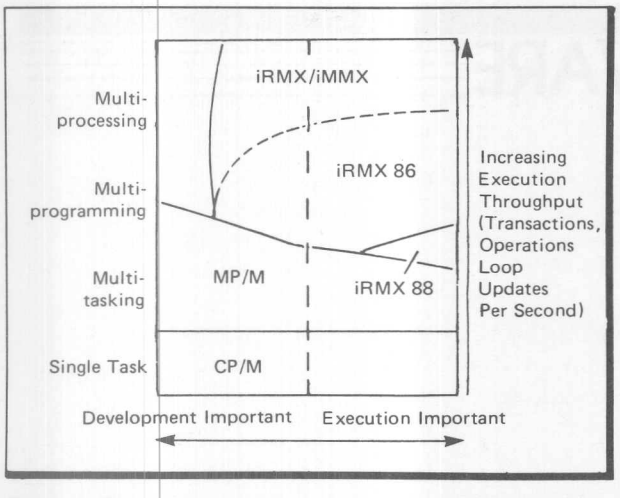


Figure 2: These microcomputer operating systems are primarily for efficient execution.

share the same resources, (e.g., memory, peripherals, I/O ports, a floating point program or the processor itself).

Although virtually all μC operating systems provide users with methods to accomplish these three main purposes, certain operating systems — designed for specific types of applications — have gone far beyond the three common objectives.

One method of viewing μC operating systems has to do with the primary purpose to which the μC is directed. Two major distinctions exist. On one hand, some μC operating systems are used primarily to develop new software systems. In this case, the operating system often includes only features and routines that are useful to persons writing and compiling software programs.

On the other hand, other μC operating systems are directed primarily at efficient execution of software programs for various applications. In this case, the operating system often includes routines and abilities targeted to fast, easy program execution.

software development OS

Within the current world of μC operating systems, examples of those oriented to efficient software development include such products as UNIX (developed by Bell Laboratories) and its related XENIX from MicroSoft, CP/M and CP/M-86 from Digital Research, and ISIS and Networked Development Systems from Intel. CP/M and ISIS-Series II are examples of operating systems designed for the technology and power of 8 bit μC systems. UNIX was designed to match the 16 bit minicomputer, (e.g., DEC PDP-11) and now is being ported (e.g., XENIX) to the newer, more powerful 16 bit μC systems.

Development oriented software systems provide growing levels of programmer support. CP/M, for example, is a single user, single terminal system. UNIX provides support to multiple terminals, so more than one person can be developing software at one time. Intel's Networked Development System provides support for both multiple users over a network, and support for development on multiple processors as well.

execution oriented OS

The second major category of operating systems comprises those primarily targeted to meet the needs of efficient execution of application software already written, developed, and tested. This category includes the largest number of systems, and is often critical to effective utilization of μCs in applications such as industrial control, communications, transaction and word processing, interactive graphics, and simulation.

Often, execution programs working in conjunction with execution-oriented operating systems are developed on a mainframe or minicomputer, with a cross-translator. The other alternative is to develop the software on a μC using a development oriented operating system that has tailored its human interface more to the trained programmer than users of the end product.

In addition to the standard purposes of an operating system — simple human interfaces, initialization, and resource management — operating systems for efficient program execution have a number of other objectives as well. These include real-time operation, multiprogramming, multitasking, multiprocessing, and effective scheduling and priority determination.

real-time operation

Many real-time operations are dedicated to specific applications, such as controlling a machine or series of machines. As such, they often run the same software programs over and over again, depending on inputs received from monitoring and measuring devices attached to the machine they control.

The primary characteristic of such applications is that the data or input that causes the μC to act is not regular, and does not occur at a particular rate. As a result, the μC program and operating system must be able to handle inputs as they occur, and to monitor or control activities based on these real-time inputs.

multiprogramming

This refers to the ability of an operating system to support several independent applications executing concurrently. If a μC, for example, is used to provide an integrated business office system, the software might be divided into a number of separate applications. One might focus on WP, another to manage the printer, and perhaps another to manage an interoffice electronic mail service. Each of these applications would involve a number of tasks devoted to different parts of the system.

An operating system that supports multiprogramming allows this division, and consequently makes it easier to develop the different applications separately, and insure that they do not interfere with each other. This is accomplished by establishing a separate environment for each application. These environments provide the basic appearance of a series of individual machines, while sharing the resources of only one. A multiprogramming operating system must manage this division, keep track of the tasks and requirements of each application, and ensure that each task is given the correct priority.

multitasking

Multitasking refers to the ability of an operating system to effectively manage several tasks, of one computer program, that are operating simultaneously. Multitasking is an important sub-set of multiprogramming, wherein several programs are concurrently being executed. Many applications require that one applications program involve different tasks. Often, these tasks must operate based on data developed in other tasks, and hence a form of task-to-task communication is required. Generally, operating systems designed for efficient program execution require some form of "executive" to manage tasks, priorities, and intertask communication. As a result, common resources such as μP memory and I/O de-

vices can be shared by multiple tasks and can be kept as busy as possible, adding to overall system efficiency.

multiprocessing

Multiprocessing refers to the ability of an operating system to support multiple processors. In certain applications, the demands of the applications exceed the capacity of a single processor or μC. As a result, more processors may be added. When this occurs, the role of the operating system is to efficiently allocate different processing requirements to the various processors, to keep track of which jobs have been sent where, and to assure that the total system resources are effectively utilized. Multiprocessing is becoming more attractive as a way to expand the functions of particular μC applications without the need to entirely rewrite a program.

As μCs become less costly compared to software development and maintenance, many systems will use multiple processors, and operating systems will be required to efficiently manage multiprocessing configurations. A means for linkage of multiple processors and operating systems is Intel's iMMX 800 (Multibus Message Exchange) software and iSBC 550 Erthernet communications controller. They support the needs of local area network applications such as office automation, distributed data processing, factory data collection, research data collection, intelligent terminal and other EDP-related applications.

scheduling and priority determination

As μC operating systems are required to manage even more complex functions — such as multiple programs, multiple

Processor Management:	iRMX 86	UNIX	CP/M	RSX-11M	RT-11	MTOS-86
Scheduling	Realtime	Multiprogram	Batch	Realtime	Realtime	Realtime
Multitasking	Yes (64K)	No	No	Yes (64K)	No	Yes (4K)
Priority Levels	255	None	None	250	None	255
Multiprogramming	Yes (64K)	Yes (64K)	No	Yes (64K)	Foreground /Background	No
Multiprocessor	iMMX	No	No	No	No	Yes (16)
Multiuser	Release 5	Yes	No	Yes (4)	No	No
Interrupt Management	Yes	No	No	Yes	Yes	Yes
Error Management	4 levels	Yes	No	Yes	Yes	No
Powerfail Protect	No	No	No	Yes	No	No
Memory Management:	**iRMX 86**	**UNIX**	**CP/M**	**RSX-11M**	**RT-11**	**MTOS-86**
Dynamic	Yes (1MB)	Yes (64K)	No	Optional	No	Yes (64K)
# of Memory Pools	64K	One	One	8	One	32
Memory Resident	Yes	No	Yes	Yes	No	Yes
Application Loader	Yes	Yes	Yes	Yes (11S: No)	Yes (RT^2: No)	No
Bootstrap Loader	Yes	Yes	No	Yes	Yes	No
(P) ROM'able	Yes	No	No	No	No	Yes
Device Management:	**iRMX 86**	**UNIX**	**CP/M**	**RSX-11M**	**RT-11**	**MTOS-86**
Concurrent I/O	Yes	Yes	No	Yes	Yes	Yes
I/O Buffering	Yes	Yes	No	Yes	Yes	No
Reentrant I/O	Yes	Yes	No	Yes	Yes	Yes
Asynchronous I/O	Yes	No	No	Yes	Yes	Yes
Synchronous I/O	Yes	Yes	Yes	Yes	Yes	No
Device Independent I/O	Yes	Yes	Yes	Yes	Yes	Yes
Max. # of Drivers	64K		256	256	16	256
Data Management:	**iRMX 86**	**UNIX**	**CP/M**	**RSX-11M**	**RT-11**	**MTOS-86**
File Support:						
1. Sequential	Yes	Yes	Yes	Yes	Yes	Yes
2. Indexed Seq.	No		No	Yes	No	No
3. Direct Access	Yes	Yes	Yes	Yes	Yes	Yes
Swapping	No	Yes	No	Yes	No	No
Overlays	Yes	Yes	Yes	Yes	Yes	No
Hierarchical Directories	Yes	Yes	No	No	No	No
Stream Files	Yes	Yes	No	No	No	No
Mailboxes	Yes	No	MP/M	No	No	No
Critical Regions	Yes	Yes	No	No	No	Yes
Host System For Development	Yes (With UDI)	Yes	Yes	Yes (11S: No)	Yes (RT^2: No)	No

Figure 3: Comparison of microcomputer operating systems

tasks, and multiple processors — the ability of the operating system to schedule activities becomes a primary consideration. In early multiprogramming operating systems, many of these scheduling routines were time driven. That is, one program would be allowed to execute for a certain time. It would then be interrupted, and another program would be allowed to execute. This time driven scheduling, in effect, forced multiprogramming to occur, but also often involved a significant amount of operating system overhead to manage the process.

Subsequently, the approach of event driven scheduling was developed. With this approach, programs and tasks are allowed to proceed until some predetermined event causes the operating system to interrupt the running task and substitute another. In many applications, event driven scheduling is the most efficient manner of allocating resources. In addition, event driven operating systems can often be modified to include some time driven scheduling routines, where the reverse is not possible. As a result, event driven systems are more flexible and can manage the μC and other system resources more efficiently.

future issues

As the μC world continues to evolve rapidly, changes in semiconductor technology will continue to force evolution and improvement in operating systems. As this evolution occurs, a number of trends and developments will influence the user's choice of appropriate operating systems. Some of these developments include:

Very Large Scale Integration (VLSI) Trends. As more complex functions are integrated into μC chips,operating systems will be required to support a broader variety of needs and application programs. The benefits of VLSI — such as increasing density, substantial improvements in function, and rapidly declining costs per function — accrue to those who rapidly use the newest technologies. As a result, μC users on the leading edge often can build significant competitive advantages by being first to market.

With regard to operating systems, trends toward greater VLSI integration imply that operating systems should be evaluated based on their ability to most rapidly use technological advances. This ability to capitalize on VLSI trends includes:

- Operating systems with the potential to be integrated into silicon. Intel, for example, has introduced a device that integrates timers, an interrupt controller, and multiprogramming and multitasking operating system "primitives" into one device. (These operating system functions are equivalent to those of the iRMX 86 kernal.) In essence, some of the traditional software functions will become part of the hardware. Such a development opens a number of vistas for future integration.
- Operating systems organized to take advantage of new trends in μC architecture. One of the most promising developments in this area is object-oriented architecture such as that implemented on iAPX 86 processors under the iRMX 86 operating system and on Intel's new iAPX-432 32 bit micromainframe product family. Essentially, an object-oriented architecture treats different kinds of data and instructions as "objects" and provides common functions that can manipulate objects in a consistent manner. Moreover, the object oriented architecture also meshes closely with the new types of high level languages — such as Ada — now being brought to market. These developments begin to provide μCs designed to optimize both program development and execution.

The benefits of VLSI — such as increasing density, substantial improvements in function, and rapidly declining costs per function — accrue to those who rapidly use the newest technologies.

Standards for μC Languages. μC applications have evolved requiring higher level languages so less technical users can easily participate in VLSI technology. Without substantial advanced planning, the use of higher level languages can cause significant problems with operating systems. One example of how advanced planning is done is Intel's approach. Intel's iRMX 86 operating system contains both a Universal Development Interface (UDI) and a Universal Runtime Interface (URI). These two interfaces provide a standard upon which high level languages can be both developed and run. In essence, the UDI/URI approach is developing a "software bus" or series of standards for easy and consistent language development. Without such an approach to standardization, each new language could be retarded in its development and each new processor could require a completely new set of compilers.

Among the many benefits of this approach are those of particular interest to persons who have been hesitant to use μCs because of the lack of standardized, high-level computer languages. With the development of the UDI/URI approach in iRMX 86, for example, languages currently or soon to be available include FORTRAN 86, PASCAL 86, PLM/86 and ASM 86.

Another major advantage of software standards now being developed is that many vendors can now create languages that will operate effectively on the same operating system. The software bus concept provides these vendors assurance their languages will operate, and it also provides μC users with a significantly broader range of application languages and even pre-developed software. Among the newer computer languages currently under development by independent software vendors as a result of the UDI/URI standardization are BASIC, COBOL, and "C".

This one development — the standardization of software development and runtime environments — has the potential to be as significant to μC users as earlier efforts to standardize on communications methods (such as the IEEE 488 standard) or μC standardization such as the Multibus (IEEE P796). Once a standard is developed and accepted, many different manufacturers can develop products with the assurance that they will be compatible with other products. Hence, the user obtains a broader selection and applications innovation is enhanced. Moreover, users need concentrate only on learning one approach, with a corresponding improvement in speed and productivity of applications efforts.

Standards for μC Networks. Increasing use of VLSI means that μC costs per function are declining. As a result, many new applications will become cost-effective. One of the major new application areas VLSI is stimulating is local area networking. The costs of μCs and supporting memory are now declining to the point where local and even global area networks make more and more sense. As networking becomes more practical, however, new approaches to networking standards will be required.

Standards are particularly important for networking μC operating systems, since the operating system will be required to manage many of the networking resources. In this regard, Intel Corp, Xerox Corp, and Digital Equipment Corp have joined together to develop Ethernet, a local area network standard. (Many other firms are also becoming involved with Ethernet, including Hewlett-Packard, Siemens, Nixdorf, Olivetti, and Zilog.)

One of Intel's Ethernet responsibilities is to provide standard modules and standard interfaces for Ethernet users. These include high level, data link layer interfaces to an Ethernet controller on the standard Multibus, a new Multibus Interprocessor Protocol (MIP), and a method to support Ethernet with the iRMX 86 operating system in conjunction with Intel's new Multibus Message Exchange (iMMX). Intel intends to provide VLSI implementations of these standards, up to the data link layer. As futher Ethernet standards evolve, the major benefit to networking μC users will be the ability to take advantage of a wide variety of products with the assurances they will work together effectively.

Protection of Software Investment. Many μC users have substantial investments in μC software and are consequently concerned that these investments do not become obsolete before having provided an adequate return on the resources invested. Many current minicomputer users, for example, would like to take advantage of the density, size, and low cost per function benefits of μCs but are hesitant to redevelop their existing application software.

Recent developments in μC software are directed toward solving this concern. The trends toward software and networking standards, for example, will provide a basis for rapid development of new language compilers, and emulators that allow minicomputer users to migrate to more technically advanced μCs without a commensurate reinvestment in software development.

Other trends in μC operating systems, such as increasing modularity and configurability, also support this direction. Intel's iRMX 86 operating system, following this trend, is developed in modules. This allows future integration of certain modules into silicon as conditions warrant. Modularity and configurability also mean users can eliminate portions of the operating system not appropriate to their application without a performance penalty. In addition, Intel's approach to operating system design means that current μC systems users can easily and cost-effectively move most of their existing software from 8 bit to 16 bit μCs as their application requirements expand. This design consideration, most noticeable in the iRMX 86 operating system, guarantees substantial user software investments are protected while users can, at the same time, rapidly take advantage of the latest developments in VLSI technology.

hardware advances mean change

Rapidly advancing developments in μC technology are causing a corresponding change in μC operating systems. More and different operating systems are becoming available as users' needs evolve and become more specialized.

The rapid march of VLSI technology also pressures manufacturers, OEMs, and end users to develop and evaluate operating systems based on their ability to directly translate VLSI advances into applications.

Moreover, the need for a series of operating systems standards is clear. Without standards such as the UDI and URI, operating system development could retard the widespread and fast growing use of productive microelectronics technology. D

ARTICLE
REPRINT

AR-196

November 1981

The iRMX™ 86 Operating System

Bruce Schafer
and Jack Davis
Intel Corporation

Order Number: 210341-001

The overall structure of the iRMX 86 Operating System . . . an O.S. designed to exploit the advantages of Intel's VLSI technology.

THE iRMX 86 OPERATING SYSTEM

Bruce Schafer and Jack Davis

INTRODUCTION

Over the past several years, microcomputers have become faster and less expensive. Accompanying this increase in raw horsepower have been enhancements in hardware architecture including high-speed floating-point co-processors, multiple processor support, and soon, Local Area Network controller chips.

This rapid increase in VLSI capability poses a question to microcomputer users: How do they keep up? Already, the cost of developing and maintaining software packages is many times more than the cost of developing the underlying hardware. Fortunately, microcomputer vendors have recognized the problem. Some of the things they can do to help include:

- —Provide a wide variety of operating system features as a set of user-selectable building blocks.
- —Provide software support for co-processors.
- —Provide software that allows the application to take advantage of multiple processors for increased application throughput.
- —Provide software that supports the use of the Ethernet* protocol.
- —Integrate operating system and hardware functions on the same VLSI component.
- —Support standard interfaces that make it easy to move application packages from one operating system to another and to take advantage of future generations of VLSI.

The iRMX 86 Operating System and supporting Intel subsystems provide these capabilities. By taking advantage of them, users can turn the "future shock" of VLSI to their advantage. As such, the iRMX 86 product can be described as a VLSI operating system.

This paper provides an overview of the features of the iRMX 86 Operating System. Associated papers describe how its capabilities are used to support multiple processors [10] and Ethernet [11, 12]. Another paper describes how many of the basic features of the operating system are being integrated with hardware functions [13].

PURPOSES OF MICROCOMPUTER OPERATING SYSTEMS

Reduced Application Investment

Because software costs are rising while hardware costs are falling, microcomputer customers have discovered that they must carefully account for their software development investment. This accounting must include both the original development and the maintenance of the software. Many microcomputer users have found that it is much cheaper to purchase software rather than develop software from scratch. An operating system allows customers to save a significant amount of time and money in developing their particular application program. By reducing their development costs, applications that would otherwise be unprofitable become profitable.

Improved Portability of Applications

Initial costs of development are not the only major concern for microcomputer customers. The cost of developing new products in an existing or new product line is also a key concern. When one microcomputer board is used to create an initial product offering, another member of the same board family is often used to extend the product line. This is true because Intel is constantly adding new features to its family of boards which offer new hardware functions and increased performance. In order to take advantage of new microcomputer components or boards, customers are interested in using as much of their existing software as possible. The use of an operating system will hide many of the details of the underlying hardware and greatly ease moving an application to a new board.

Maximizing Hardware Utilization

Many applications allow monitoring and controlling more than one device. An operating system can make it appear to the application programmer that he has more than one processor at his disposal. This is accomplished by allowing more than one activity to occur asynchronously. The operating system can go further in providing mechanisms for communication and synchronization between programs running on the microcomputer.

Support of Sound Development Methodology

Modular construction is a key way to control the cost of software development and maintenance. Each module ideally "hides" or encapsulates the effect of major decisions

*Ethernet is a Trademark of Xerox Corporation.

such as how data is represented. Similar to structured programming, the modular design process permits a complex design to be partitioned into a structure of cooperating modules. This both facilitates top-down development and simplifies the maintenance and evolution of large software systems. Even though users of microcomputer systems are usually familiar with these concepts, the lack of effective tools and tight development schedules often force them to take a choice between modular construction and quick implementation. High level programming languages have played a key role in allowing customers to obtain the best of both worlds. An operating system can take this one step further by adding facilities to simplify modular implementation. For instance, it can allow the customer to identify separate asynchronous activities that his applications must accomplish and write a separate program for each of these activities. The operating system can allow these separate programs to be run as separate tasks and communicate with each other as necessary. When a particular part of the application is to be modified, the change can be isolated to one or a very few number of the original application programs. When additional functions need to be added, additional tasks can be added to the system with little or no change to the existing tasks.

COMMON FEATURES OF MICROCOMPUTER OPERATING SYSTEMS

The varied purposes of microcomputer operating systems have led to some relatively standard features. Following are some of the features provided by the iRMX 86 Operating System.

Multitasking and Multiprogramming

Based on the goals summarized above, a key role of a microcomputer operating system is to allow for multiple programs to execute on the same processor. This maximizes the utilization of the hardware and at the same time provides for modular construction of applications. Most importantly, it allows the application programmer to manage the complexity inherent in real-time applications where multiple asynchronous events are occurring.

When several programs must execute concurrently (or at least appear to do so), an operating system can provide multitasking. Each executing program is called a task. When these tasks must execute in separate environments, the operating system can support multiprogramming to provide these environments: The set of tasks executing in a separate environment can then be called a job.

Interrupt Mapping

Since most real-time operating systems are event driven, they must use the interrupt structure of the underlying hardware. The operating system can provide the mechanism that transforms the hardware interrupts into events that will cause particular tasks to execute and service the interrupt. In this way each task need be only aware of the interrupt that it is managing.

Timer Support

Real-time applications often require the use of the hardware timer provided by the microcomputer. This may be because they must be aware of elapsed time in order to accomplish their purpose. Another reason is that external events may not occur when they should. The software must be aware that they have not occurred and take corrective action. In either case the programs running as separate tasks may each need a separate timer.

Memory Allocation

Many applications cannot predict their actual memory usage ahead of time. This is because they must deal with an unpredictable environment. A key part of this environment is the operator who invokes various functions of the application. The parameters provided by the operator often affect how much memory the program needs. Besides the operator, the external devices connected to the microcomputer generate a variety of events to which the software must respond immediately. These events are not entirely predictable and thus the amount of memory required to respond to and control these events is not predictable. By providing for dynamic memory allocation, an operating system can help an application adapt to its unpredictable environment without statically allocating the worst-case memory requirements to each part of the application.

Device Support

Often a significant portion of the application development time is spent writing complex code that interfaces the application to vendor-supplied devices. An operating system can provide software that does this interfacing. This feature reduces the customer's development cost and elapsed time.

File Support

Operating systems, in general, use random access devices such as disk drives or magnetic tapes to store and retrieve information. The very simplest of these applications might only store one set of data on each device. Far more common than this, however, is the situation where one disk is to store many different kinds of information. Examples of this information may be parametric information that affects the activity of the application, intermediate data required by the application to accomplish its purpose, and data that the application generates which will be used later.

An operating system must manage the secondary storage space represented by the random access device. This management will usually include support for dynamic allocation of random access space, automatic bookkeeping of this space, and naming each portion used by the application. In

this way the application can treat a single random access unit as if it were many separate devices each of which can be randomly accessed.

Loading Support

Many microcomputer applications involve no mass storage devices and thus all of the operating system and application code resides in read-only memory. For systems that include mass storage devices, this allows some of the code to be loaded into read/write memory and provides some significant advantages to the programmer. These include

(1) Software can be updated and distributed on disk or similar media.

(2) If not all parts of the applications must execute concurrently, less total memory may be required if only the code required is loaded into memory.

Human Interface

The vast majority of microcomputer applications involve some interface to a human operator. Many of these applications use standard cathode-ray-tube terminals in order to accomplish this interface. An operating system can reduce the cost of using this interface in three ways:

1) By providing for common editing functions that allow the human operator to correct his input before it is seen by the application.

2) By providing for the automatic invocation of the correct application program based on input by the human operator.

3) By providing functions that make it easy for the application program to determine which parameters have been specified by the operator.

This completes a general description of the features that are provided by iRMX 86. A detailed description of the functional capabilities of this operating system follows.

MAJOR FEATURES OF THE iRMX 86 OPERATING SYSTEM

In order to provide operating system support for applications using the 8086 microprocessor, Intel has developed the iRMX 86 Operating System.[1] Because the system is modular, it allows customers to choose only those features of the operating system that are needed by their applications.

Nucleus

The iRMX 86 Nucleus provides for multitasking, interrupt control, timer support, and intertask communication.[2] While many of these concepts are the same as in other microcomputer operating systems the application interface is quite different. One reason for this is that iRMX 86 interfaces encapsulate the details of the implementation so that it can be more easily changed without affecting the application. This encapsulation is accomplished by implementing each mechanism as a type manager. Each type manager provides a set of objects that are defined by their attributes and the operations that can be performed by them. The basic object types supported are task, segment, mailbox, semaphore, region, job, and extension. One of these objects, the task, also serves as the subject in the sense that tasks are the active elements of the system and perform operations on all objects.

Tasks. All operations are performed by tasks. A task is an executing program and is characterized by a set of processor register values, a priority, a containing job, and a dispatcher state.

Segments. A segment is a contiguous portion of memory described by a base and a length in bytes. The base of a segment can be loaded into a segment register for use as a code segment, stack segment, data segment, or extra segment.

Mailboxes. Mailbox objects are used by a task when it wishes to pass an object to another task in the system. A set of tasks can use this mailbox to implement synchronization, mutual exclusion, and communication.

Semaphores. Semaphores are used in the place of mailboxes where no actual information needs to be communicated between the tasks. Semaphores have the advantage of requiring less execution time overhead and thus may be a sound alternative where performance is critical. They are also useful in solving resource allocation problems, especially when the number of units of the resource is large or if it is desirable to allocate several units at once.

Regions. The region object type is used to implement critical regions via mutual exclusion. Regions have the advantage of preventing the deletion or suspension of a task during a critical operation. They are also used to guarantee that a high priority task will not wait an excessive amount of time for a resource held by a lower priority task that is currently in a region. This is accomplished by raising the effective priority of the task holding the critical region whenever a higher priority task is waiting for it.

Jobs. Job objects represent the environment in which a set of tasks can operate. This environment is limited by the resources given to an application. Several different things can make it desirable to divide an application into multiple environments:

1) Control Dynamic Memory Allocation.

When multiple applications are implemented on a single microprocessor the extent to which each application can

account for the other applications' memory needs is limited. By providing separate memory allocation environments for each application, the user can allocate portions of the total memory to each. In this way he can ensure that one application will not allocate memory to the disadvantage of others. This approach also makes it easier to avoid a key hazard of dynamic allocation, the possibility of memory deadlock.

2) Separately Invoke & Abort Individual Applications

While many applications only require a static set of tasks, some applications involve the dynamic invocation of individual subapplications and require the ability to abort these subapplications at any time. By providing separate environments, the iRMX 86 Operating System allows an individual subapplication to be aborted and have its resources returned to the system without affecting other subapplications in the system.

3) Provide Separate Name Spaces

When multiple applications are run on the same microcomputer, these applications are often designed and implemented by separate programming teams. When these teams select names for external devices that are to be used by their applications, they take the risk of choosing names also used by other applications. By providing a separate environment for each executing application, the names used for each application can be kept separate. At execution time the operator can match the individual names used by the application against the resources provided by the operating system. A particular example of this concept occurs when one program is invoked more than one time concurrently. During each invocation the operator can match a set of devices that the application uses against a particular subset of the devices available. In this way the same program can access several sets of devices, at the same time, because from the point of view of the operating system the program is running in separate environments.

Extensions. The final basic object type is the extension object.[9] The extension object is used by operating extensions to create new types. New objects of the new type can be created as a composite of existing objects. Operators are also provided for deleting a composite object and for obtaining the component objects of a composite object.

The concept of restricting access to objects is an integral part of the iRMX 86 model. When a task creates an object all tasks in its job are automatically given access to the object. A task can pass an object to a task in another job. Two mechanisms for communicating access to an object have been built into the iRMX 86 Nucleus: object directories and mailboxes. The advantage of object directories is that they allow a task to obtain access to an object by knowing only its name. The advantage of mailboxes is that they also can be used for synchronization and communication between tasks.

To increase the ease in which programs can be debugged, each iRMX 86 function returns an exception code. This code indicates whether the requested operation was successfully performed, and if not, what went wrong. These exception codes may be handled either by instructions immediately following the call to the operating system or by a separate error handler. In the latter case, an error condition will automatically cause control to transfer to a user-provided routine or, by default, to one provided by the system.

Terminal Handler

The Terminal Handler provides a real-time, asynchronous interface between a terminal and tasks running under the supervision of the Nucleus.[3] It can be used either with or without the Debugger. The Terminal Handler provides the following features:

- Line editing
- Multicharacter type-ahead
- Control characters for suspending and resuming output at the terminal
- A means of awakening the Debugger.

The Terminal Handler can be accessed either directly under the supervision of the Nucleus or through the Basic I/O System described later.

Debugger

The Debugger is designed specifically for debugging and monitoring systems running under the supervision of the Nucleus.[4] A special Debugger is very helpful in debugging such systems because their real-time and multi-tasking characteristics render useless many ordinary debugging techniques. The iRMX 86 Debugger is sensitive to the data structures used by the Nucleus and can give "snapshots" of tasks at critical moments. It can also be used to alter the contents of memory. If desired, the Debugger can be included in a debugged application system for trouble-shooting in the field. If it is included the Debugger requires only the support of the Nucleus and the Terminal Handler.

Basic I/O System

The iRMX 86 Basic I/O System provides facilities for accessing devices and files residing on random access devices.[5] By taking a modular approach, it allows the customer to choose between support for physical devices such as terminals and random access devices, and sophisticated support for files on the random access devices. It accomplished this goal by providing two types of drivers.

The first are device drivers. Device drivers allow the customer to choose from the set of Intel-provided device drivers in order to support the controllers that are being used. In addition, the customer can add his own device drivers to match custom device controllers.

The second type of driver is called a file driver. The file driver accomplishes goals similar to a device driver in that it is a modular piece of the I/O System which allows the customer just those facilities needed by his application. File drivers implement different types of files.

Named files. The most general type of file provided by the iRMX 86 Basic I/O System is that of a named file. A named file is a byte-oriented random-access file which is given a name to identify it among those on a particular volume. Files can serve as both data files and directories. Directories can point to both data files and directories. In this way a file can be designated by a sequence of file names from the root or main directory on a volume through a sequence of directory files to the actual file being designated. Once located, a file can be opened and closed as many times as desired without further directory searches. This type of file naming mechanism is called a hierarchical file structure.

The accessing of the individual data blocks of a file is designed to both minimize allocation of space on a volume and to minimize the number of disk accesses required to access an arbitrary location in a file. For small files an arbitrary data block can be read with a single physical seek-and-read operation. In large files this can be accomplished with at most, two seek-read combinations. Because there is inevitably a trade-off between space and performance, the Basic I/O System allows the application to specify to what extent space should be compromised for performance or vice versa.

Physical files. The second major type of file provided by the Basic I/O System is the physical file type. Physical files are accessed similiarly to physical devices. In order to provide a common interface to a wide variety of devices the interfaces to physical files assumes that any byte on a device can be accessed. The device driver for a particular device then chooses to what extent it supports this file. For instance, a device driver for a line printer would return an error upon a request for seek or read.

Stream files. A third type of file provided by the Basic I/O System is called stream files. A stream file is a sequence of bytes. A task can add bytes to the end of this sequence of bytes and/or read bytes from the front of the sequence of bytes. Any bytes read are consumed by the read operation and thus can only be read once. A key use of string files is to allow a program that normally writes to a physical device or to a disk file to direct its output to another program.

Because the Basic I/O System provides three basic file drivers which are accessed via a common interface, application programmers do not have to be concerned whether the data that is being output is being written to a physical device, a data file, or directly to another program.

Extended I/O System

The iRMX 86 Extended I/O System[6] builds upon the facilities provided by the Basic I/O System and provides the additional facilities to accomplish two general goals:

1) decreasing the cost of implementing applications that use the facilities of the Basic I/O System.

2) providing additional features not found in the Basic I/O System.

The fact that the Basic I/O System is designed to be very general means that some of its system calls are overly complex when used in simple situations. The Extended I/O System provides an optional interface that allows calling sequences to be greatly simplified when some of the more sophisticated features of the Basic I/O System are not needed.

One of the additional facilities provided by the Extended I/O System is the notion of logical names. Logical names allow applications to refer to devices without using the actual physical names of the devices. This allows an application to be written for a standard set of devices and then be executed with a variety of different devices. This accounts for both the fact that the user of an application program will vary over time and the fact that the set of available devices will change over time. The Extended I/O System also extends the concept of logical names to include directory files and data files. In this way an application program can refer to a device location without knowing whether it is using an entire physical device, a directory on that device, or a particular data file on that device. Consequently a data file can be substituted for a device like a line printer or a directory on a large disk can be substituted for a smaller disk.

The Extended I/O System provides support for automatic buffering as an optional facility. This support accomplishes two goals.

1) Allow the physical accesses to the devices to match its physical characteristics. In particular, it allows the number of bytes requested of the device to match the characteristics of the device such as its sector size.

2) Allow the physical access of the device to be concurrent with the execution of the application program. In this way the throughput of the system can be increased by overlapping CPU and I/O time.

The first goal is accomplished by having the Extended I/O System allocate a buffer whose size matches the physical characteristics of the device. Input and output requests are accomplished using the buffer or buffers as intermediate storage. In this way the actual request made to the device controller matches the controller, and is independent of what the application has requested.

The second goal is accomplished by providing one or two buffers where data can be moved to and from at the same time the application is executing. To use the example of sequential input, when the file is open, the Extended I/O System can initiate reads into the buffers that it has allocated for the application. When the application makes its first request, the data it needs may already be in one of those buffers and can be returned immediately to the application. By the time the data in the first buffer is exhausted the second buffer may have been filled. While the second buffer is being used the Extended I/O System can refill the first. In this way, assuming that the execution time required to process a buffer full of information is comparable to the time required to read from the disk, the application will run concurrently with the physical reading of the device. The same approach can be used for sequential output, in this case the physical writing is delayed until the buffer is filled.

By combining the notions of automatic buffer size and automatic overlap, the total throughput of the system can be increased and the execution time of a particular application can be decreased. The Extended I/O System provides both these facilities in a way that makes them appear as if the application is doing a simple sequence of reads and writes. It completely hides the buffering and the overlap algorithm from the application.

Application Loader

The Application Loader uses the I/O System that to load object files into memory[7]. With the loader, you can store some of your code on disk and load it into memory only when you actually need it. This can lower the memory requirements for your application system.

The Application Loader accepts the following types of files:

1) *Absolute Files:* The loader places absolute code into memory at predetermined locations.

2) *Load Time Locatable (LTL):* These files contain code which the loader can assign to any available memory in the job's memory pool. This version of the loader will automatically update instructions as they are loaded to account for the fact that in a multisegment program the code in one segment may refer to code and data in other segments. Since the loader is responsible for allocating the segments, it can update the code with the appropriate base values while it is loading the code.

The Application Loader also supports overlays. This allows an application to be constructed as a "root" and a set of overlays. This significantly reduces the amount of memory required to support large applications.

Human Interface

The iRMX 86 Operating System provides an additional layer of software called the Human Interface.[8] This layer makes it particularly easy for customers to add customer facilities to the system. It is designed to provide support for interactive commands whose code is usually not resident in memory. In addition to this goal it provides a standard set of commands for the manipulation of files.

In order to make it easy to add custom commands to the system, the Human Interface provides for automatically loading and invoking the appropriate program based on the commands entered by the operator. Once a program is loaded and invoked in this way, it can access its parameters by making a series of calls to the Human Interface routines. These routines will return connections to files as well as other parameters.

Rather than require each customer to implement his own set of basic commands, the Human Interface provides a standard set of file manipulation commands. This set of commands includes commands for renaming files, copying files, displaying a list of the files in a particular directory, creating files, changing access to files, and deleting files.

Bootstrap Loader

The iRMX 86 Bootstrap Loader[7] is used to load the iRMX 86 system and/or application programs into memory from mass storage and begin their execution. It consists of two stages.

The first stage provides a rudimentary device driver and uses it to read in the first part of the second stage. In addition, the first stage may provide a file name to the second stage to identify which version of the system is to be loaded. The first stage may also provide for automatic or manual bootstrap device selection.

The second stage reads the rest of itself in and then finds and loads the operating system. Using the device driver from the first stage, the second stage interprets the file structure on the disk. Since the first stage and device driver handle all the device dependent matters, the same second stage can be used on all iRMX 86 disks regardless of the type of device used. Separately located modules may be combined in a

library, so that the various layers of the operating system and the application may be loaded from one file.

SUMMARY

This paper has outlined the advantages of vendor-supplied operating systems software for microcomputers. Although these advantages parallel those used for operating systems in minicomputers and mainframe computers, the specific facilities required by a microcomputer customer are often different. For example, many microcomputer applications do not require operating system support for devices and files. The iRMX 86 Operating System answers this need by providing layered software. It is organized around a basic Nucleus that supports multitasking and offers I/O facilities as optional layers above this Nucleus. A customer can choose how many of these facilities he requires and then add his application software on top of the operating system.

By taking this approach iRMX 86 reduces the investment required by the customer, improves the portability of applications from one microcomputer to another, allows one microcomputer to be used for multiple concurrent applications, and provides a model that makes it much easier to change and add features.

CONCLUSION

The dramatic increase of computing power provided by microcomputers represents a challenge. How can this power be harnessed without turning the entire labor force into computer programmers? Part of the answer is provided by vendor-supplied operating systems.

By taking advantage of the availability of both high-performance microcomputer hardware and off-the-shelf software, designers can leverage their expertise and investment. End-users can quickly and effectively put microcomputers to use in their applications. Original Equipment Manufacturers can take advantage of what they know best, their market and their technology. In this way, they can play a *leadership* role in taking the microcomputer revolution to their marketplace.

REFERENCES

1. *Introduction to the iRMX 86 Operating System,* Intel Corporation, 1980.
2. *iRMX 86 Nucleus Reference Manual,* Intel Corporation, 1981.
3. *iRMX 86 Terminal Handler Reference Manual,* Intel Corporation, 1981.
4. *iRMX 86 Debugger Reference Manual,* Intel Corporation, 1981.
5. *iRMX 86 Basic I/O System Reference Manual,* Intel Corporation, 1981.
6. *iRMX 86 Extended I/O System Reference Manual,* Intel Corporation, 1981.
7. *iRMX 86 Loader Reference Manual,* Intel Corporation, 1981.
8. *iRMX 86 Human Interface Reference Manual,* Intel Corporation, 1981.
9. *iRMX 86 System Programmer's Reference Manual,* Intel Corporation, 1981.
10. Ronald M. Smith, Multiple Processor Support with the iRMX 86 Operating System, *Proceedings of the iRMX 86 Technical Symposium,* Intel Corporation, 1981.
11. Jack Inman, Getting onto Ethernet with the iRMX 86 Operating System, *Proceedings of the iRMX 86 Technical Symposium,* Intel Corporation, 1981.
12. Ted Forgeron, Intel OEM Building Blocks Support Natural Addition of Ethernet Capabilities, *Proceedings of the iRMX 86 Technical Symposium,* Intel Corporation, 1981.
13. Phillip L. Barrett, VLSI Operating Systems, *Proceedings of the iRMX 86 Technical Symposium,* Intel Corporation, 1981.